9_Multi_agentes

April 9, 2025

Computación Inteligente:

Grandes Modelos de Lenguajes

Clase 9: Multi-Agentes LLM

En términos técnicos, un agente de IA es una entidad de software diseñada para realizar tareas de forma autónoma o semiautónoma en nombre de un usuario u otro programa. Estos agentes utilizan la inteligencia artificial para tomar decisiones, realizar acciones e interactuar con su entorno u otros sistemas. Algunas de las características clave de los agentes son las siguientes:

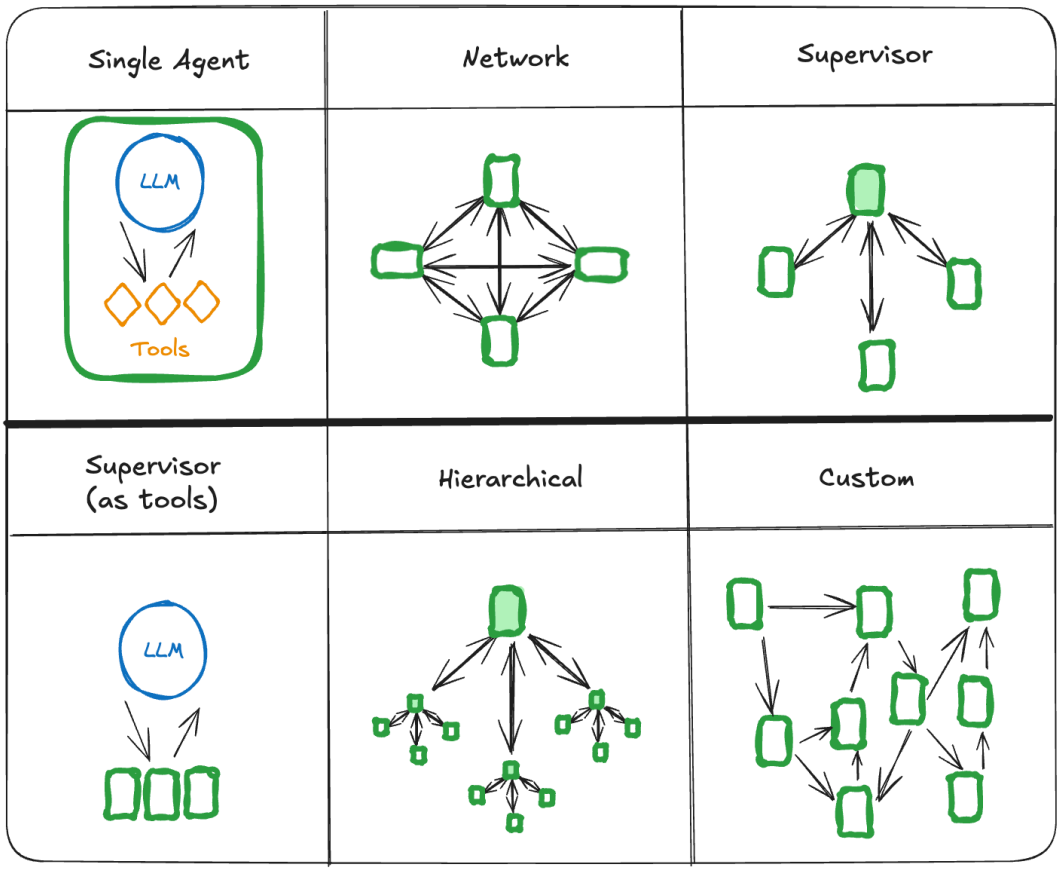
- **Autonomía**: Los agentes de IA operan sin intervención humana constante. Pueden realizar tareas de forma independiente una vez que se les asigna un objetivo.
- **Toma de decisiones**: Utilizan algoritmos, reglas y modelos de IA para tomar decisiones basadas en sus percepciones y objetivos. Esto incluye evaluar diferentes opciones y seleccionar la mejor estrategia.
- **Aprendizaje**: Muchos agentes de IA incorporan técnicas de aprendizaje automático para mejorar su rendimiento con el tiempo. Pueden aprender de experiencias pasadas y adaptarse a nuevas situaciones.
- **Interacción**: Los agentes de IA pueden comunicarse y colaborar con usuarios, otros agentes o sistemas. Esta interacción puede implicar el procesamiento del lenguaje natural, el envío y la recepción de datos o la realización de tareas coordinadas.
- **Especialización**: Los agentes de IA pueden especializarse para tareas o dominios específicos. Por ejemplo, algunos agentes pueden estar diseñados para la navegación web, mientras que otros pueden gestionar interacciones con bases de datos, realizar cálculos complejos o generar imágenes.
- **Orientado a objetivos**: Los agentes de IA suelen programarse con objetivos específicos.

Trabajan para alcanzarlos mediante una secuencia de acciones y decisiones. En resumen, los agentes de IA son herramientas potentes que pueden automatizar y optimizar una amplia gama de actividades, desde tareas simples y repetitivas hasta escenarios complejos de resolución de problemas, lo que los hace invaluables en diversas aplicaciones e industrias.

Imagine aprovechar todos los conceptos anteriores integrados y trabajar juntos para alcanzar objetivos predefinidos y lograr los resultados deseados. Estas tareas podrían ejecutarse en un proceso secuencial o jerárquico, con todos los agentes trabajando como un equipo coordinado. Esta potente colaboración puede revolucionar la forma en que abordamos problemas complejos, haciendo que los procesos sean más eficientes y los resultados más efectivos. Aquí es donde entra en escena el marco CrewAI.

0.1 Arquitecturas de los Sistemas Multiagentes

Hay varias maneras de conectar agentes en un sistema multiagente:



- **Red:** cada agente puede comunicarse con todos los demás. Cualquier agente puede decidir a qué agente llamar a continuación.
- **Supervisor:** cada agente se comunica con un único agente supervisor. El agente supervisor decide a qué agente se debe llamar a continuación.
- **Supervisor (invocación de herramientas):** este es un caso especial de arquitectura de supervisores. Los agentes individuales pueden representarse como herramientas. En este caso, un agente supervisor utiliza un LLM de invocación de herramientas para decidir a qué herramientas del agente llamar, así como los argumentos que se les pasarán.
- **Jerárquico:** se puede definir un sistema multiagente con un supervisor de supervisores. Esta es una generalización de la arquitectura de supervisores y permite flujos de control más complejos.
- **Flujo de trabajo multiagente personalizado:** cada agente se comunica solo con un subconjunto de agentes. Partes del flujo son deterministas, y solo algunos agentes pueden decidir a qué otros agentes llamar a continuación.

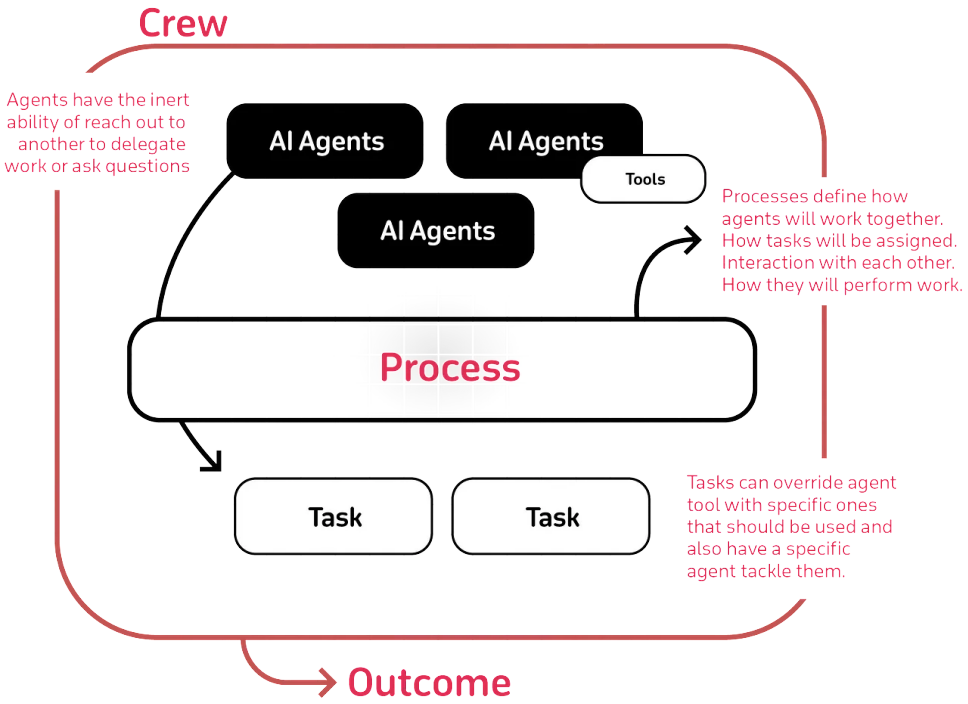
1 ¿Qué es CrewAI?

CrewAI es un framework de Python ágil y ultrarrápido, creado completamente desde cero, completamente independiente de LangChain u otros frameworks de agentes. Ofrece a los desarrolladores simplicidad de alto nivel y un control preciso de bajo nivel, ideal para crear agentes de IA autónomos adaptados a cualquier escenario.

- **CrewAI Crews**: Optimizados para la autonomía y la inteligencia colaborativa.
- **CrewAI Flows**: Habilita un control granular basado en eventos, llamadas LLM individuales para una orquestación precisa de tareas y soporte nativo para Crews..

1.1 Conceptos básicos — CrewAI

- **Agentes**: Son unidades independientes programadas para realizar tareas, tomar decisiones y comunicarse con otros agentes. Pueden utilizar herramientas, que pueden ser simples funciones de búsqueda o integraciones complejas que involucran otras cadenas, API, etc.
- **Tareas**: Las tareas son asignaciones o trabajos que un agente de IA debe completar. Pueden incluir información adicional, como qué agente debe realizar la tarea y qué herramientas podría necesitar.
- **Tripulación**: Una tripulación es un equipo de agentes, cada uno con un rol específico, que trabajan juntos para lograr un objetivo común. El proceso de formación de una tripulación implica reunir a los agentes, definir sus tareas y establecer una secuencia de ejecución.



1.1.1 Usaremos como modelo GEMINI 2.5 PRO

De forma predeterminada, los modelos de OpenAI se utilizan como llm en CrewAI. Para obtener el máximo rendimiento con CrewAI, considere usar GPT-4 o el modelo ligeramente más económico

GPT-4o-mini de OpenAI para sus agentes de IA. Estos modelos son la base de sus agentes y tienen un impacto significativo en sus capacidades.

En nuestro caso usaremos Gemini 2.5 PRO. Que se puede usar gratis.

[1]:
```
import os
from dotenv import load_dotenv
from langchain_google_genai import ChatGoogleGenerativeAI
from crewai import Agent, Task, Crew, Process

load_dotenv()

# Load the google gemini api key
google_api_key = os.getenv("GEMINI_API_KEY")

# Set gemini pro as llm
llm = ChatGoogleGenerativeAI(
    model="models/gemini-2.5-pro-exp-03-25", verbose=True, temperature=0.9,␣
 ↪google_api_key=google_api_key
)
```

[2]:
```
from openai import OpenAI

client = OpenAI(
  api_key=google_api_key,
  base_url="https://generativelanguage.googleapis.com/v1beta/openai/"
)

model = client.models.retrieve("gemini-2.5-pro-exp-03-25")
llm = model.id
```

[4]:
```
llm =llm.replace('models', 'gemini')
llm
```

[4]:
```
'gemini/gemini-2.5-pro-exp-03-25'
```

1.2 Crear agentes para planificar, redactar y editar el contenido de un artículo.

Un agente es una unidad autónoma programada para:

- Realizar tareas.
- Tomar decisiones.
- Comunicarse con otros agentes.

1.2.1 Atributos del Agente

- **Rol**: Define la función del agente dentro del equipo. Determina el tipo de tareas para las que es más adecuado.

- **Goal**: El objetivo individual que el agente busca alcanzar. Guía su proceso de toma de decisiones.
- **Backstory**: Proporciona contexto al rol y el objetivo del agente, enriqueciendo la interacción y la dinámica de colaboración.
- **LLM**: (opcional) Representa el modelo de lenguaje que ejecutará el agente. Obtiene dinámicamente el nombre del modelo de la variable de entorno OPENAI_MODEL_NAME; el valor predeterminado es "gpt-4" si no se especifica.
- **Tools**: (opcional) Conjunto de capacidades o funciones que el agente puede usar para realizar tareas. Se espera que sean instancias de clases personalizadas compatibles con el entorno de ejecución del agente. Las herramientas se inicializan con el valor predeterminado de una lista vacía.
- **Function Calling LLM**: (opcional) Especifica el modelo de lenguaje que gestionará la llamada a la herramienta para este agente, anulando la función del equipo que llama a LLM si se pasa. El valor predeterminado es Ninguno.
- **Max Iter**: (opcional) El número máximo de iteraciones que el agente puede realizar antes de ser forzado a proporcionar su mejor respuesta. El valor predeterminado es 25.
- **Max RPM**: (opcional) El número máximo de solicitudes por minuto que el agente puede realizar para evitar los límites de velocidad. Es opcional y puede dejarse sin especificar; el valor predeterminado es Ninguno.
- **max_execution_time**: (opcional) Tiempo máximo de ejecución para que un agente ejecute una tarea. Es opcional y puede dejarse sin especificar; el valor predeterminado es Ninguno, lo que significa que no hay tiempo máximo de ejecución.
- **Verbose**: (opcional) Al establecerlo en Verdadero, se configura el registrador interno para proporcionar registros de ejecución detallados, lo que facilita la depuración y la monitorización. El valor predeterminado es Falso.
- **Allow Delegation**: (opcional) Los agentes pueden delegar tareas o preguntas entre sí, garantizando que cada tarea sea gestionada por el agente más adecuado. El valor predeterminado es Verdadero.
- **Step Callback**: (opcional) Una función que se llama después de cada paso del agente. Esto puede usarse para registrar las acciones del agente o para realizar otras operaciones. Sobrescribirá la `step_callback` del equipo.
- **Cache**: (opcional) Indica si el agente debe usar caché para el uso de herramientas. El valor predeterminado es `True`.

1.2.2 Agente planificador de contenido

[5]:
```
planificador = Agent(
    role="Planificador de Contenido",
    goal="Planifique contenido atractivo y preciso sobre {tema}",
    backstory="Estás trabajando en la planificación de un articulo de "
              "opinión sobre el tema: {tema} para una publicación. "
              "Recopilas información que ayuda "
```

```
            "a la audiencia a aprender algo "
            "y a tomar decisiones informadas. "
            "Debes preparar un esquema detallado  "
            "y los temas y subtemas relevantes que deben "
            "formar parte del artículo."
            "Tu trabajo es la base para que el redactor "
            "de contenido escriba un artículo sobre este tema.",
    llm=llm,
    allow_delegation=False,
    verbose=True
)
```

1.2.3 Agente escritor de contenido

[6]:
```
escritor = Agent(
    role="Escritor de Contenido",
    goal="Escribe un artículo de opinión perspicaz  "
         "y con hechos precisos sobre el tema: {tema}",
    backstory="Estás trabajando en un nuevo artículo "
              "de opinión sobre el tema: {tema}. "
              "Basas tu escritura en el trabajo del "
              "Planificador de Contenido, quien proporciona un esquema "
              "y contexto relevante sobre el tema. "
              "Sigues los objetivos principales y  "
              "la dirección del esquema, "
              "según lo establecido por el Planificador de Contenido. "
              "También proporcionas perspectivas objetivas  "
              "e imparciales y las respaldas con información proporcionada "
              "por el Planificador de Contenido. "
              "Usted reconoce en su artículo de opinión "
              "cuando sus declaraciones son opiniones "
              "y no declaraciones objetivas.",
    allow_delegation=False,
    llm=llm,
    verbose=True
)
```

1.2.4 Agente editor de contenido

[7]:
```
editor = Agent(
    role="Editor",
    goal="Editar el articulo determinado para alinearlo "
         "con el estilo de escritura de una publicación informativa. ",
    backstory="Eres un editor que recibe un artículo "
              "del redactor de contenido. "
              "Su objetivo es revisar el artículo "
              "para asegurarse de que siga las mejores prácticas de "
```

```
            "publicaciones, proporcione puntos de vista "
            "equilibrados al brindar opiniones o afirmaciones y "
            "también evite temas u opiniones controvertidos "
            "mportantes cuando sea posible.",
    llm=llm,
    allow_delegation=False,
    verbose=True
)
```

1.2.5 Crear tarea

Las tareas dentro de crewAI pueden ser colaborativas, lo que requiere la colaboración de varios agentes. Esto se gestiona mediante las propiedades de la tarea y está orquestado por el proceso de CrewAI, lo que mejora el trabajo en equipo y la eficiencia.

Atributos de la tarea

- **Description**: Una descripción clara y concisa de lo que implica la tarea.
- **Agent**: El agente responsable de la tarea, asignado directamente o por el proceso de CrewAI.
- **Expected Output**: Una descripción detallada de cómo se ve la finalización de la tarea.
- **Tools**: (opcional) Las funciones o capacidades que el agente puede utilizar para realizar la tarea.
- **Async Execution**: (opcional) Si se configura, la tarea se ejecuta de forma asíncrona, lo que permite avanzar sin esperar a que se complete.
- **Context**: (opcional) Especifica las tareas cuyas salidas se utilizan como contexto para esta tarea.
- **Config**: (opcional) Detalles de configuración adicionales para el agente que ejecuta la tarea, lo que permite una mayor personalización.
- **Output JSON:** (opcional) Genera un objeto JSON; requiere un cliente OpenAI. Solo se puede configurar un formato de salida: (opcional) Genera un objeto JSON; requiere un cliente OpenAI. Solo se puede configurar un formato de salida.
- **Output Pydantic**: (opcional) Genera un objeto de modelo Pydantic que requiere un cliente OpenAI. Solo se puede configurar un formato de salida.
- **Output File**: (opcional) Guarda la salida de la tarea en un archivo. Si se usa con Salida JSON o Salida Pydantic, especifica cómo se guarda la salida.
- **Call**: (opcional) Un objeto invocable de Python que se ejecuta con la salida de la tarea al finalizar.
- **Human Input**: (opcional) Indica si la tarea requiere retroalimentación humana al final, útil para tareas que requieren supervisión humana.

1.2.6 Crear tarea de planificación

[8]:
```
plan = Task(
    description=(
        "1. Prioriza las últimas tendencias, los actores clave "
        "y las noticias destacadas sobre {tema}.\n"
        "2. Identifica al público objetivo, "
        "considerando sus intereses y dificultades.\n"
        "3. Desarrolla un esquema de contenido detallado que incluya "
        "una introducción, puntos clave y una llamada a la acción.\n"
        "4. Incluye palabras clave SEO y datos o fuentes relevantes."
    ),
    expected_output="Un documento de plan de contenido completo "
        "con un esquema, análisis de audiencia, "
        "palabras clave de SEO y recursos.",
    agent=planificador,
)
```

1.2.7 Crear tarea de escritura

[9]:
```
escribir = Task(
    description=(
        "1. Utilice el plan de contenido para crear una publicación "
            "de blog atractiva sobre {tema}.\n"
        "2. Incorporar palabras clave SEO de forma natural.\n"
        "3. Las secciones/subtítulos están nombrados de forma "
            "adecuada y atractiva.\n"
        "4. Asegúrese de que la publicación esté estructurada "
            "con una introducción atractiva, un cuerpo perspicaz "
            "y una conclusión resumida.\n"
        "5. Corrección de errores gramaticales y "
            "alineación con la voz de la marca.\n"
    ),
    expected_output="Una publicación bien escrita  "
        "en formato Markdown, lista para publicar, "
        "cada sección debe tener 2 o 3 párrafos.",
    agent=escritor,
)
```

1.2.8 Crear tarea de edición

[10]:
```
edicion = Task(
    description=("Revise la publicación para detectar "
                 "errores gramaticales y "
                 "asegurar que esté alineada con la voz de la marca."),
    expected_output="Una publicación bien escrita  "
                    "en formato Markdown, lista para publicar, "
                    "cada sección debe tener 2 o 3 párrafos.",
```

```
    agent=editor
)
```

1.2.9 Creando la tripulación

- Crea tu tripulación de agentes.
- Transmite las tareas que realizarán esos agentes.
- Nota: En este ejemplo sencillo, las tareas se ejecutarán secuencialmente (es decir, son interdependientes), por lo que el orden de las tareas en la lista es importante.
- `verbose=2` permite ver todos los registros de la ejecución.

[11]:
```
crew = Crew(
    agents=[planificador, escritor, editor],
    tasks=[plan, escribir, edicion],
    verbose=True
)
```

1.2.10 Ejecutar

[12]:
```
entradas = {"tema":"Estudio comparativo de PydanticAI, LangGraph, Autogen y␣
 ↪Crewai para construir sistemas multiagente"}
resultado = crew.kickoff(inputs=entradas)
```

```
                    Crew Execution Started
```

```
 Crew Execution Started
 Name: crew
 ID: bcb5e39a-b461-4410-9480-e7e3023a1de1
```

```
Crew: crew
  Task: 1c62c7a3-2adb-4906-bfd7-a4e190a50ccb
    Status: Executing Task…
```

```
Crew: crew
   Task: 1c62c7a3-2adb-4906-bfd7-a4e190a50ccb
      Status: Executing Task…
       Agent: Planificador de Contenido
           Status: In Progress
```

```
# Agent: Planificador de Contenido
## Task: 1. Prioriza las últimas tendencias, los actores clave y
las noticias destacadas sobre Estudio comparativo de PydanticAI, LangGraph,
Autogen y Crewai para construir sistemas multiagente.
2. Identifica al público objetivo, considerando sus intereses y dificultades.
3. Desarrolla un esquema de contenido detallado que incluya una introducción,
puntos clave y una llamada a la acción.
4. Incluye palabras clave SEO y datos o fuentes relevantes.
```

```
Crew: crew
   Task: 1c62c7a3-2adb-4906-bfd7-a4e190a50ccb
      Status: Executing Task…
       Agent: Planificador de Contenido
           Status: In Progress
           Thinking…
```

```
Crew: crew
   Task: 1c62c7a3-2adb-4906-bfd7-a4e190a50ccb
      Status: Executing Task…
       Agent: Planificador de Contenido
           Status: In Progress
```

```
# Agent: Planificador de Contenido
```

Final Answer:

Documento de Plan de Contenido: Estudio Comparativo de Frameworks Multiagente

1. Priorización de Tendencias, Actores Clave y Noticias:

*   **Tendencias Clave:**
    *   **Auge de Sistemas Multiagente (SMA):** Creciente interés en utilizar múltiples agentes de IA especializados que colaboran para resolver tareas complejas que un solo modelo monolítico no puede abordar eficientemente.
    *   **Orquestación y Colaboración:** El foco está en cómo gestionar la comunicación, el flujo de tareas y el estado compartido entre agentes. Frameworks como LangGraph y CrewAI abordan esto directamente.
    *   **Agentes Autónomos y Toma de Decisiones:** Desarrollo de agentes capaces de planificar, ejecutar y auto-corregir tareas con mínima intervención humana (Autogen es fuerte aquí).
    *   **Estructuración de Datos y Fiabilidad:** La necesidad de que los agentes interactúen con herramientas y API de manera fiable impulsa el uso de bibliotecas como Pydantic para la validación y estructuración de entradas/salidas (relevancia de Pydantic en el ecosistema).
    *   **Estado y Memoria:** Gestionar el estado a lo largo de interacciones complejas y ciclos es un desafío clave (LangGraph destaca en esto).
    *   **Human-in-the-loop:** Incorporar la supervisión y la intervención humana de forma fluida en los flujos de trabajo de los agentes (Autogen tiene mecanismos para esto).
*   **Actores Clave:**
    *   **LangChain Team:** Desarrolladores de LangGraph, parte del ecosistema LangChain más amplio.
    *   **Microsoft Research:** Impulsores de Autogen, con un fuerte enfoque en la investigación y la experimentación.
    *   **CrewAI Developers:** Equipo enfocado en un framework específico para colaboración basada en roles.
    *   **Pydantic Team:** Mantenedores de la biblioteca Pydantic, fundamental para la validación de datos en Python, ampliamente utilizada en el ecosistema de IA.
    *   **Comunidad Open Source:** Contribuyentes, usuarios y desarrolladores que crean tutoriales, extensiones y comparten casos de uso para todos estos frameworks.

```
Crew: crew
   Task: 1c62c7a3-2adb-4906-bfd7-a4e190a50ccb
      Status: Executing Task…
       Agent: Planificador de Contenido
           Status:   Completed



Crew: crew
   Task: 1c62c7a3-2adb-4906-bfd7-a4e190a50ccb
      Assigned to: Planificador de Contenido
      Status:   Completed
       Agent: Planificador de Contenido
           Status:   Completed

                          Task Completion
                                                                                 ␣
↪
 Task Completed                                                                  ␣
↪
 Name: 1c62c7a3-2adb-4906-bfd7-a4e190a50ccb                                      ␣
↪
 Agent: Planificador de Contenido                                               ␣
↪
                                                                                 ␣
↪
                                                                                 ␣
↪



Crew: crew
   Task: 1c62c7a3-2adb-4906-bfd7-a4e190a50ccb
      Assigned to: Planificador de Contenido
      Status:   Completed
       Agent: Planificador de Contenido
           Status:   Completed
   Task: 5d3d9b09-1b8f-48a4-9033-429a3f5cd9c9
      Status: Executing Task…
```

```
Crew: crew
   Task: 1c62c7a3-2adb-4906-bfd7-a4e190a50ccb
      Assigned to: Planificador de Contenido
      Status:   Completed
       Agent: Planificador de Contenido
           Status:   Completed
   Task: 5d3d9b09-1b8f-48a4-9033-429a3f5cd9c9
      Status: Executing Task…
       Agent: Escritor de Contenido
           Status: In Progress
```

```
# Agent: Escritor de Contenido
## Task: 1. Utilice el plan de contenido para crear una
publicación de blog atractiva sobre Estudio comparativo de PydanticAI,
LangGraph, Autogen y Crewai para construir sistemas multiagente.
2. Incorporar palabras clave SEO de forma natural.
3. Las secciones/subtítulos están nombrados de forma adecuada y atractiva.
4. Asegúrese de que la publicación esté estructurada con una introducción
atractiva, un cuerpo perspicaz y una conclusión resumida.
5. Corrección de errores gramaticales y alineación con la voz de la marca.
```

```
Crew: crew
   Task: 1c62c7a3-2adb-4906-bfd7-a4e190a50ccb
      Assigned to: Planificador de Contenido
      Status:   Completed
       Agent: Planificador de Contenido
           Status:   Completed
   Task: 5d3d9b09-1b8f-48a4-9033-429a3f5cd9c9
      Status: Executing Task…
       Agent: Escritor de Contenido
           Status: In Progress
           Thinking…
```

```
Crew: crew
```

```
Task: 1c62c7a3-2adb-4906-bfd7-a4e190a50ccb
  Assigned to: Planificador de Contenido
  Status:   Completed
   Agent: Planificador de Contenido
       Status:   Completed
Task: 5d3d9b09-1b8f-48a4-9033-429a3f5cd9c9
   Status: Executing Task...
    Agent: Escritor de Contenido
        Status: In Progress
```

```
# Agent: Escritor de Contenido
```

Final Answer:

```markdown
# Opinión: PydanticAI, LangGraph, Autogen vs. CrewAI: Eligiendo el Framework Correcto para tus Sistemas Multiagente

El desarrollo de la Inteligencia Artificial avanza a pasos agigantados, y una de las fronteras más emocionantes es la de los **Sistemas Multiagente (SMA)**. Estos sistemas, compuestos por múltiples agentes de IA especializados que colaboran, prometen resolver tareas de una complejidad inabordable para modelos monolíticos. Hemos pasado de la teoría a la práctica, y hoy presenciamos una explosión de frameworks diseñados específicamente para construir, orquestar y gestionar estos equipos de agentes digitales.

Sin embargo, esta abundancia de herramientas presenta un nuevo desafío para desarrolladores, ingenieros de IA y arquitectos de soluciones: ¿Cómo elegir el framework adecuado? La decisión no es trivial, ya que cada opción viene con su propia filosofía, arquitectura, fortalezas y debilidades. Equivocarse en la elección puede llevar a callejones sin salida, complejidades innecesarias o sistemas que no escalan como se esperaba.

Este artículo de opinión busca arrojar luz sobre este panorama, ofreciendo una **comparativa basada en el análisis de Pydantic (en su rol dentro de la IA, a menudo referido informalmente como "PydanticAI"), LangGraph, Autogen y CrewAI**. No pretendo tener la verdad absoluta, sino compartir una perspectiva informada para ayudarte a navegar estas opciones. Desglosaremos brevemente cada contendiente: **Pydantic**, fundamental para la estructura y validación de datos; **LangGraph**, para construir grafos de estado complejos dentro del ecosistema LangChain; **Autogen**, enfocado en conversaciones colaborativas entre agentes; y **CrewAI**, diseñado para orquestar equipos basados en roles y procesos. Al final, tendrás una mejor idea de qué herramienta podría encajar mejor en tus próximos proyectos de **agentes autónomos**.

## La Columna Vertebral Invisible: El Rol Crucial de Pydantic en la IA

Antes de sumergirnos en los frameworks de orquestación, es fundamental aclarar el papel de Pydantic. A menudo se menciona "PydanticAI", pero es más preciso hablar del uso de **Pydantic *en* aplicaciones de IA**. Pydantic es una biblioteca Python estándar de oro para la validación y el análisis de datos. Su poder reside en el uso de type hints de Python para definir esquemas de datos claros, concisos y robustos.

En el contexto de los **Sistemas Multiagente**, Pydantic se vuelve
```

```
Crew: crew
   Task: 1c62c7a3-2adb-4906-bfd7-a4e190a50ccb
      Assigned to: Planificador de Contenido
      Status:   Completed
       Agent: Planificador de Contenido
            Status:   Completed
   Task: 5d3d9b09-1b8f-48a4-9033-429a3f5cd9c9
      Status: Executing Task…
       Agent: Escritor de Contenido
            Status:   Completed
```

```
Crew: crew
   Task: 1c62c7a3-2adb-4906-bfd7-a4e190a50ccb
      Assigned to: Planificador de Contenido
      Status:   Completed
       Agent: Planificador de Contenido
            Status:   Completed
   Task: 5d3d9b09-1b8f-48a4-9033-429a3f5cd9c9
      Assigned to: Escritor de Contenido
      Status:   Completed
       Agent: Escritor de Contenido
            Status:   Completed
```

```
                              Task Completion
                                                                                          ␣
↪
 Task Completed                                                                           ␣
↪
 Name: 5d3d9b09-1b8f-48a4-9033-429a3f5cd9c9                                               ␣
↪
 Agent: Escritor de Contenido                                                             ␣
↪
                                                                                          ␣
↪
                                                                                          ␣
↪
```

```
Crew: crew
   Task: 1c62c7a3-2adb-4906-bfd7-a4e190a50ccb
      Assigned to: Planificador de Contenido
      Status:   Completed
       Agent: Planificador de Contenido
           Status:   Completed
   Task: 5d3d9b09-1b8f-48a4-9033-429a3f5cd9c9
      Assigned to: Escritor de Contenido
      Status:   Completed
       Agent: Escritor de Contenido
           Status:   Completed
   Task: 388617b7-070e-4840-bea7-2208cb926777
      Status: Executing Task…
```

```
Crew: crew
   Task: 1c62c7a3-2adb-4906-bfd7-a4e190a50ccb
      Assigned to: Planificador de Contenido
      Status:   Completed
       Agent: Planificador de Contenido
           Status:   Completed
   Task: 5d3d9b09-1b8f-48a4-9033-429a3f5cd9c9
      Assigned to: Escritor de Contenido
      Status:   Completed
       Agent: Escritor de Contenido
           Status:   Completed
   Task: 388617b7-070e-4840-bea7-2208cb926777
      Status: Executing Task…
       Agent: Editor
           Status: In Progress
```

```
# Agent: Editor
## Task: Revise la publicación para detectar errores gramaticales
y asegurar que esté alineada con la voz de la marca.
```

```
Crew: crew
   Task: 1c62c7a3-2adb-4906-bfd7-a4e190a50ccb
      Assigned to: Planificador de Contenido
      Status:   Completed
       Agent: Planificador de Contenido
           Status:   Completed
   Task: 5d3d9b09-1b8f-48a4-9033-429a3f5cd9c9
```

```
      Assigned to: Escritor de Contenido
      Status:   Completed
       Agent: Escritor de Contenido
           Status:   Completed
   Task: 388617b7-070e-4840-bea7-2208cb926777
      Status: Executing Task…
       Agent: Editor
           Status: In Progress
           Thinking…



 Crew: crew
   Task: 1c62c7a3-2adb-4906-bfd7-a4e190a50ccb
      Assigned to: Planificador de Contenido
      Status:   Completed
       Agent: Planificador de Contenido
           Status:   Completed
   Task: 5d3d9b09-1b8f-48a4-9033-429a3f5cd9c9
      Assigned to: Escritor de Contenido
      Status:   Completed
       Agent: Escritor de Contenido
           Status:   Completed
   Task: 388617b7-070e-4840-bea7-2208cb926777
      Status: Executing Task…
       Agent: Editor
           Status: In Progress



# Agent: Editor
```

Final Answer:

```markdown
# Análisis Comparativo: Pydantic, LangGraph, Autogen y CrewAI en Sistemas Multiagente

El desarrollo de la Inteligencia Artificial avanza a pasos agigantados, y una de las fronteras más interesantes es la de los **Sistemas Multiagente (SMA)**. Estos sistemas, compuestos por múltiples agentes de IA especializados que colaboran, ofrecen un potencial significativo para resolver tareas de una complejidad que puede resultar inabordable para modelos monolíticos. Hemos pasado de la teoría a la práctica, y hoy observamos una creciente disponibilidad de frameworks diseñados específicamente para construir, orquestar y gestionar estos equipos de agentes digitales.

Esta abundancia de herramientas, sin embargo, presenta un desafío para desarrolladores, ingenieros de IA y arquitectos de soluciones: la elección del framework adecuado. La decisión no es trivial, ya que cada opción presenta su propia filosofía, arquitectura, fortalezas y debilidades. Seleccionar una herramienta que no se alinee bien con las necesidades del proyecto puede llevar a complejidades innecesarias o sistemas que no escalan como se esperaba.

Este artículo ofrece un **análisis comparativo de Pydantic (en su rol dentro de la IA), LangGraph, Autogen y CrewAI**, buscando aportar claridad a este panorama. El objetivo es ofrecer una perspectiva informada para ayudar a navegar estas opciones. Desglosaremos brevemente cada contendiente: **Pydantic**, fundamental para la estructura y validación de datos; **LangGraph**, para construir grafos de estado complejos dentro del ecosistema LangChain; **Autogen**, enfocado en conversaciones colaborativas entre agentes; y **CrewAI**, diseñado para orquestar equipos basados en roles y procesos. Al finalizar, el lector tendrá una mejor comprensión de qué herramienta podría encajar mejor en sus próximos proyectos de **agentes autónomos**.

## La Columna Vertebral Invisible: El Rol Crucial de Pydantic en la IA

Antes de profundizar en los frameworks de orquestación, es fundamental aclarar el papel de Pydantic. Aunque a veces se menciona "PydanticAI", es más preciso hablar del uso de **Pydantic *en* aplicaciones de IA**. Pydantic es una biblioteca Python ampliamente reconocida para la validación y el análisis de datos. Su principal ventaja radica en el uso de type hints de Python para definir esquemas de datos claros, concisos y robustos, lo que mejora la calidad y fiabilidad del código.
```

```
Crew: crew
   Task: 1c62c7a3-2adb-4906-bfd7-a4e190a50ccb
      Assigned to: Planificador de Contenido
      Status:   Completed
       Agent: Planificador de Contenido
           Status:   Completed
   Task: 5d3d9b09-1b8f-48a4-9033-429a3f5cd9c9
      Assigned to: Escritor de Contenido
      Status:   Completed
       Agent: Escritor de Contenido
           Status:   Completed
   Task: 388617b7-070e-4840-bea7-2208cb926777
      Status: Executing Task…
       Agent: Editor
           Status:   Completed
```

```
Crew: crew
   Task: 1c62c7a3-2adb-4906-bfd7-a4e190a50ccb
      Assigned to: Planificador de Contenido
      Status:   Completed
       Agent: Planificador de Contenido
           Status:   Completed
   Task: 5d3d9b09-1b8f-48a4-9033-429a3f5cd9c9
      Assigned to: Escritor de Contenido
      Status:   Completed
       Agent: Escritor de Contenido
           Status:   Completed
   Task: 388617b7-070e-4840-bea7-2208cb926777
      Assigned to: Editor
      Status:   Completed
       Agent: Editor
           Status:   Completed


                        Task Completion

 Task Completed
 Name: 388617b7-070e-4840-bea7-2208cb926777
```

```
 Agent: Editor
↪
↪
↪
↪


                        Crew Completion
↪
↪
 Crew Execution Completed
↪
 Name: crew
↪
 ID: bcb5e39a-b461-4410-9480-e7e3023a1de1
↪
↪
↪
```

2 Análisis Comparativo: Pydantic, LangGraph, Autogen y CrewAI en Sistemas Multiagente

El desarrollo de la Inteligencia Artificial avanza a pasos agigantados, y una de las fronteras más interesantes es la de los **Sistemas Multiagente (SMA)**. Estos sistemas, compuestos por múltiples agentes de IA especializados que colaboran, ofrecen un potencial significativo para resolver tareas de una complejidad que puede resultar inabordable para modelos monolíticos. Hemos pasado de la teoría a la práctica, y hoy observamos una creciente disponibilidad de frameworks diseñados específicamente para construir, orquestar y gestionar estos equipos de agentes digitales.

Esta abundancia de herramientas, sin embargo, presenta un desafío para desarrolladores, ingenieros de IA y arquitectos de soluciones: la elección del framework adecuado. La decisión no es trivial, ya que cada opción presenta su propia filosofía, arquitectura, fortalezas y debilidades. Seleccionar una herramienta que no se alinee bien con las necesidades del proyecto puede llevar a complejidades innecesarias o sistemas que no escalan como se esperaba.

Este artículo ofrece un **análisis comparativo de Pydantic (en su rol dentro de la IA), LangGraph, Autogen y CrewAI**, buscando aportar claridad a este panorama. El objetivo es

ofrecer una perspectiva informada para ayudar a navegar estas opciones. Desglosaremos brevemente cada contendiente: **Pydantic**, fundamental para la estructura y validación de datos; **LangGraph**, para construir grafos de estado complejos dentro del ecosistema LangChain; **Autogen**, enfocado en conversaciones colaborativas entre agentes; y **CrewAI**, diseñado para orquestar equipos basados en roles y procesos. Al finalizar, el lector tendrá una mejor comprensión de qué herramienta podría encajar mejor en sus próximos proyectos de **agentes autónomos**.

2.1 La Columna Vertebral Invisible: El Rol Crucial de Pydantic en la IA

Antes de profundizar en los frameworks de orquestación, es fundamental aclarar el papel de Pydantic. Aunque a veces se menciona "PydanticAI", es más preciso hablar del uso de **Pydantic *en* aplicaciones de IA**. Pydantic es una biblioteca Python ampliamente reconocida para la validación y el análisis de datos. Su principal ventaja radica en el uso de type hints de Python para definir esquemas de datos claros, concisos y robustos, lo que mejora la calidad y fiabilidad del código.

En el contexto de los **Sistemas Multiagente**, Pydantic juega un papel crucial. La comunicación fiable es la base de la colaboración efectiva entre agentes. Estos necesitan intercambiar información estructurada, interactuar con herramientas externas (APIs, bases de datos) y procesar salidas de Modelos de Lenguaje Grandes (LLMs). Pydantic permite definir "contratos" de datos explícitos: qué formato espera una herramienta, qué estructura debe tener la salida de un agente, qué parámetros son válidos para una función. Esta práctica reduce significativamente los errores en tiempo de ejecución, facilita la depuración y ayuda a garantizar que las interacciones sean predecibles y consistentes.

Es importante entender que Pydantic, por sí solo, *no* es un framework de orquestación de agentes. No gestiona el flujo de control, el estado conversacional ni la asignación de tareas entre agentes. Sin embargo, actúa como un habilitador esencial *dentro* de frameworks como LangGraph, Autogen y CrewAI, proporcionando la estructura necesaria para que estos sistemas más complejos funcionen de manera fiable. Sus fortalezas son la robustez, la claridad y su profunda integración en el ecosistema Python. Su limitación, en el contexto de esta comparativa, es que no aborda directamente la orquestación multiagente; es una pieza fundamental del puzzle, no el puzzle completo.

2.2 LangGraph: Tejiendo Agentes con Grafos de Estado

Proveniente del ecosistema **LangChain**, **LangGraph** adopta un enfoque potente y flexible para construir aplicaciones de agentes: modelarlas como grafos de estado (StateGraphs). La filosofía central es que las interacciones complejas, especialmente aquellas que involucran ciclos, memoria y toma de decisiones basada en estados previos, pueden representarse de manera natural como un grafo donde los nodos son funciones (que pueden encapsular agentes o llamadas a herramientas) y las aristas representan las transiciones entre estos estados.

La arquitectura de LangGraph se basa en conceptos claros: un estado compartido que persiste y se actualiza a lo largo del flujo, nodos que modifican este estado y aristas condicionales que dirigen el flujo basado en el estado actual. Esto lo hace particularmente adecuado para manejar la **gestión explícita y robusta del estado**, uno de los desafíos reconocidos en **sistemas multiagente** complejos. Su integración nativa con LangChain (LCEL, herramientas, modelos) es una ventaja significativa para equipos ya invertidos en ese ecosistema, permitiendo reutilizar componentes y conocimientos existentes.

Sin embargo, esta flexibilidad puede corresponder a una curva de aprendizaje potencialmente más

pronunciada en comparación con frameworks más prescriptivos. Definir los grafos, gestionar las transiciones y manejar el estado explícitamente puede requerir más código de configuración inicial ("boilerplate"), especialmente para flujos de trabajo más simples o estrictamente lineales. La abstracción del grafo, aunque potente, podría no ser la más intuitiva para todos los casos de uso o desarrolladores. LangGraph resulta ideal para procesos iterativos, agentes que necesitan replanificar dinámicamente, aplicaciones con requisitos de estado complejos y aquellos que buscan un control granular sobre el flujo de ejecución dentro del entorno **LangChain**.

2.3 Autogen: Orquestando Conversaciones Inteligentes entre Agentes

Desarrollado por Microsoft Research, **Autogen** se centra en un paradigma diferente: la **colaboración multiagente a través de conversaciones**. Su filosofía es permitir que múltiples agentes especializados interactúen entre sí, deleguen tareas y resuelvan problemas complejos de forma colaborativa, a menudo con la capacidad de ejecutar código o utilizar herramientas para lograr sus objetivos. Introduce conceptos clave como `ConversableAgent` (la unidad básica de agente), `UserProxyAgent` (que actúa como un proxy para la intervención humana) y `GroupChatManager` para gestionar patrones de conversación más complejos.

Una de las fortalezas notables de **Autogen** es su capacidad para simular dinámicas de equipo y su flexibilidad en la configuración de patrones de conversación (jerárquicos, broadcast, etc.). La integración del **human-in-the-loop** a través del `UserProxyAgent` permite una supervisión e intervención fluidas cuando sea necesario. Su fuerte respaldo de investigación y su capacidad inherente para la ejecución de código y el uso de herramientas lo hacen muy potente para tareas de resolución de problemas, como la generación y depuración de código, el análisis de datos o la escritura colaborativa.

No obstante, la configuración y personalización de estos sistemas conversacionales en Autogen puede volverse compleja a medida que aumenta la sofisticación del sistema. La gestión del estado, aunque programable, podría percibirse como menos explícita o estructurada que en LangGraph para ciertos patrones cíclicos o de larga duración. Al ser un proyecto de investigación activa, la documentación puede, en ocasiones, ir un poco por detrás de las últimas funcionalidades, lo que podría influir en la curva de aprendizaje inicial. **Microsoft Autogen** destaca en escenarios de investigación, resolución colaborativa de problemas complejos, simulación de equipos y tareas donde la dinámica conversacional y la autonomía del agente son aspectos clave.

2.4 CrewAI: Definiendo Roles y Procesos para Equipos de Agentes

CrewAI adopta lo que podría considerarse un enfoque de más alto nivel y más prescriptivo para la **orquestación IA**. Su filosofía se centra en facilitar la creación de "equipos" (Crews) de agentes autónomos, cada uno con roles claramente definidos (`role`), metas (`goal`), historias de fondo (`backstory`), tareas específicas (`Task`s) y herramientas (`tools`) asignadas. La colaboración se estructura a través de procesos predefinidos (`Process`), que pueden ser secuenciales o jerárquicos. Este modelo conceptual resulta bastante intuitivo, ya que se asemeja a cómo los equipos humanos suelen organizarse para abordar proyectos.

La principal fortaleza de **CrewAI** radica en esta abstracción basada en roles y procesos. Puede simplificar la definición de responsabilidades y la estructuración de flujos de trabajo colaborativos. Definir un `Agent`, asignarle `Task`s dentro de un `Crew` que sigue un `Process` es relativamente directo, lo que puede acelerar el prototipado y desarrollo para casos de uso bien definidos. Su curva de

aprendizaje a menudo se considera potencialmente más suave, especialmente para aquellos que buscan aplicar la **automatización con IA** en procesos de negocio estructurados o tareas colaborativas paso a paso.

Por otro lado, esta abstracción de alto nivel puede implicar ciertas limitaciones. **CrewAI** podría ofrecer menos flexibilidad que LangGraph o Autogen al implementar flujos de trabajo muy dinámicos, altamente cíclicos o que requieran patrones de comunicación no estructurados o emergentes. Al ser un framework relativamente más joven, su ecosistema y comunidad están aún en fase de crecimiento activo en comparación con alternativas más establecidas. Ofrece un control menos granular sobre las interacciones de bajo nivel entre agentes. Es una opción valiosa para la automatización de procesos de negocio (marketing, ventas, reporting), la creación de equipos de agentes con roles claros (investigador, escritor, crítico) y el prototipado rápido de ideas de **colaboración agentes IA** siguiendo flujos definidos.

2.5 Análisis Comparativo Directo: Características Clave

Para visualizar mejor las diferencias clave, la siguiente tabla resume las características principales, seguida de una discusión sobre criterios importantes:

Criterio	Pydantic (Rol Habilitador)	LangGraph	Autogen	CrewAI
Enfoque Principal	Validación/Estructura Datos	Grafos de Estado	Conversación entre Agentes	Roles y Procesos
Flexibilidad	N/A (Componente)	Muy Alta	Alta	Media-Alta (en su paradigma)
Gestión de Estado	N/A	Explícita y Robusta	Implícita (conversación) / Prog.	Gestionada por el Proceso
Curva de Aprendizaje	Baja (si se conoce Pydantic)	Media-Alta	Media-Alta	Baja-Media
Nivel Abstracción	N/A	Baja-Media	Media	Alta
Ideal para Ciclos	N/A	Sí	Posible	Menos Natural
Human-in-the-loop	N/A	Programable	Integrado (UserProxyAgent)	Posible, menos directo
Ecosistema	Python	LangChain	Propio / Python	Propio / Python
Madurez	Muy Alta (Pydantic Lib)	Media	Media-Alta (Investigación)	Baja-Media

Facilidad de Uso vs. Flexibilidad: Este es quizás uno de los trade-offs más evidentes. CrewAI prioriza una abstracción de alto nivel y una potencial facilidad de uso para flujos estructurados, lo que puede implicar menor flexibilidad inherente. LangGraph y Autogen ofrecen una flexibilidad considerablemente mayor, pero esto puede venir acompañado de una curva de aprendizaje potencialmente más pronunciada y una configuración más detallada. Análisis sugieren que CrewAI podría ofrecer un inicio más rápido para tareas basadas en roles, mientras que LangGraph y Autogen proporcionan más potencia para escenarios no convencionales o altamente personalizados.

Manejo del Estado: LangGraph destaca en esta área con su gestión explícita a través del `StateGraph`. Autogen maneja el estado principalmente a través del historial de conversaciones y la lógica programada dentro de los agentes. CrewAI abstrae gran parte de la gestión del estado dentro de la ejecución del proceso definido. La elección aquí depende del nivel de control granular que se necesite sobre el estado del sistema multiagente a lo largo de su ejecución.

Depuración y Observabilidad: Este es un desafío general en el desarrollo de SMA. LangGraph se beneficia de la integración con herramientas como LangSmith para trazabilidad. Autogen y CrewAI dependen más de logging estándar y las herramientas de depuración de Python, aunque las comunidades respectivas están trabajando activamente en mejorar la observabilidad. Este factor es crítico a medida que los sistemas crecen en complejidad y se acercan a entornos de producción.

2.6 Guía para la Toma de Decisiones: ¿Cuál Elegir y Cuándo?

La elección del framework correcto depende crucialmente de las necesidades específicas y el contexto de cada proyecto. A continuación, se presenta una guía basada en el análisis anterior:

- **Prioridad en Fiabilidad y Estructura de Datos:** Utilizar **Pydantic** (o modelos de datos similares) para definir entradas y salidas es altamente recomendable, independientemente del framework de orquestación elegido. Proporciona la base para interacciones robustas entre agentes y herramientas.
- **Prioridad en Flujos Complejos, Cíclicos y Control Granular del Estado (especialmente con uso previo de LangChain): LangGraph** emerge como un candidato fuerte. Su flexibilidad y gestión explícita del estado son adecuadas para procesos iterativos y arquitecturas complejas.
- **Prioridad en Simulación de Conversaciones Complejas, Investigación y Flexibilidad Conversacional con fácil integración humana: Autogen** resulta extremadamente potente. Es excelente para explorar la colaboración entre agentes, tareas de resolución de problemas no estructuradas y aprovechar la ejecución de código.
- **Prioridad en Modelado Rápido de Equipos con Roles Claros, Procesos Definidos y Curva de Aprendizaje más Suave para Flujos Estructurados: CrewAI** ofrece una abstracción intuitiva y eficiente. Es una buena opción para automatizar procesos de negocio y prototipar rápidamente equipos de agentes colaborativos.

Además, es útil considerar estos factores del proyecto al tomar la decisión: * **Complejidad del Flujo de Trabajo:** ¿Es principalmente lineal, cíclico, dinámico, o fuertemente conversacional? * **Experiencia del Equipo:** ¿Cuál es el nivel de familiaridad con Python avanzado, conceptos de grafos, y paradigmas de IA específicos? * **Integración con LangChain:** ¿Es un requisito o una ventaja deseable aprovechar el ecosistema LangChain? * **Requerimientos de Estado y Memoria:** ¿Son las necesidades de estado simples o complejas? ¿Se requiere memoria a corto o largo plazo? * **Importancia del Human-in-the-loop:** ¿Se necesita intervención humana frecuente y debe ser fácil de implementar? * **Madurez Requerida:** ¿Se está construyendo un prototipo exploratorio o una aplicación destinada a producción?

2.7 El Horizonte de los Frameworks Multiagente

El campo de los **frameworks para sistemas multiagente** está en plena efervescencia y evoluciona a gran velocidad. Es probable que veamos una cierta convergencia de características, donde los frameworks adopten enfoques exitosos de otros, buscando un equilibrio entre flexibilidad y facilidad

de uso. Se esperan mejoras significativas en las herramientas de depuración y observabilidad, cruciales para la adopción a escala y en entornos de producción.

También podrían surgir estándares o patrones más definidos para la comunicación entre agentes, facilitando la interoperabilidad entre diferentes sistemas o componentes. La integración con plataformas MLOps más amplias será clave para gestionar el ciclo de vida completo de estas aplicaciones complejas. En medio de esta evolución, la necesidad de una estructura de datos robusta y una validación fiable (el rol fundamental de bibliotecas como **Pydantic**) seguirá siendo un pilar esencial para construir sistemas fiables.

2.8 Conclusión: No Hay Bala de Plata, Solo la Herramienta Adecuada

Hemos explorado el panorama de cuatro actores relevantes en el espacio de los **sistemas multiagente**: Pydantic como pilar fundamental de datos, LangGraph para grafos de estado flexibles, Autogen para conversaciones colaborativas y CrewAI para equipos basados en roles y procesos. Cada uno presenta una filosofía distintiva, puntos fuertes particulares y casos de uso donde tiende a destacar.

Como suele ocurrir en el ámbito tecnológico, no existe una única herramienta "mejor" universalmente. La elección más acertada depende intrínsecamente de los requisitos del proyecto, la experiencia del equipo de desarrollo y los objetivos específicos que se persiguen. El estado actual del arte es prometedor; estas herramientas, aunque algunas relativamente jóvenes, ya permiten construir aplicaciones de IA con un grado notable de complejidad y utilidad. El potencial de los **agentes autónomos** colaborando para resolver problemas es considerable.

La recomendación final es la experimentación. Explorar los repositorios, ejecutar los ejemplos y participar en las comunidades asociadas a estos frameworks es a menudo la mejor manera de entender cuál resuena más con las necesidades particulares de un proyecto. El futuro de la IA colaborativa se está construyendo activamente, y estas herramientas son facilitadores clave en ese proceso.

¡Únete a la Conversación!

- ¿Qué framework para **sistemas multiagente** estás utilizando o planeas probar? ¿Cuáles han sido tus experiencias, desafíos o éxitos? **Comparte tus perspectivas en los comentarios.**
- Sigue nuestra publicación para más análisis y discusiones sobre **tecnología IA** y LLMs.
- Explora la documentación oficial para profundizar:
  - Pydantic
  - LangGraph
  - Autogen
  - CrewAI
- Considera unirte a las comunidades de Discord o GitHub Discussions de estos proyectos para conectar y aprender de otros desarrolladores. "'